Recitation #5

18-649 Embedded System Engineering

Friday 26-Sept-2014

Electrical & Computer ENGINEERING

Carnegie Mellon

Note: Course slides shamelessly stolen from lecture

Changes To Non-project Items

- **I apologize for the “no changes to door control” email**

- **Changes must be in the issue log**
  - Helps TAs find where changes are when things don’t match the template
- **Changes must be peer reviewed**
  - Helps you not burn yourself by making bad changes

DoorControl Traceability

- "The DoorControl never makes the doors reverse. How do we make it trace to our sequence diagrams?"

- Options:
  - Change sequence diagrams
    - Fix your sequence diagrams to match the DoorController behavior
  - Change the behavioral requirements
    - Make the DoorControl behave as described

Announcements and Administrative Stuff

- **Project 5 posted**

- **Project 5 is due Thursday Oct. 2nd by 10pm**
  - Get started if you haven't already!
  - Testing will take a while

- **.xls/.xlsx files**
  - We must be able to open them. If we can't open them, they don't exist
  - Test them on the lab computers. If we can't get our machines to open them, we will open them there

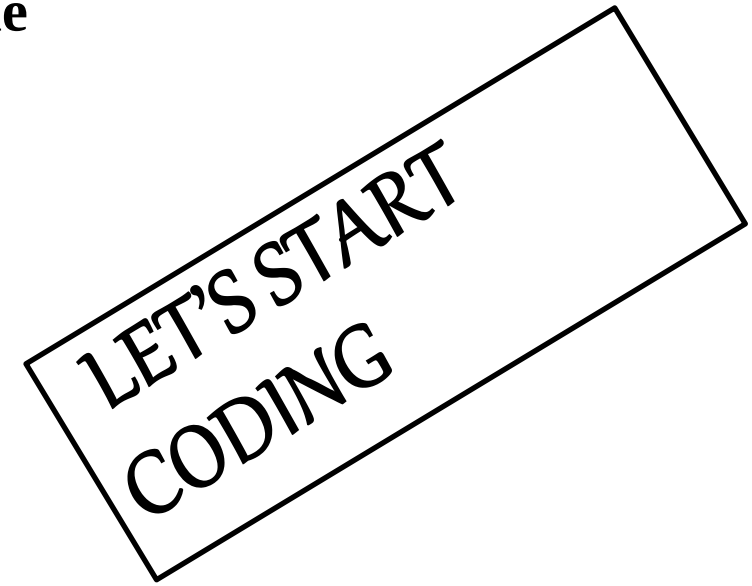
E-mail Check-list (On Admin Page)

- **Before writing that e-mail**
  - Check blackboard to see if an answer has been posted
  - Re-read the assignment to make sure you are reading it correctly
  - Look at the grading checklist to see if it has relevant information
  - Look at the Pepsi machine example to see if it provides a reasonable example
  - Discuss the problem with your teammates and see if you can agree upon a reasonable way to proceed without violating written assignment requirements

- **Regarding e-mail on assignments**
  - ***If you simply don't understand, then skip the e-mail and go to office hours***
  - If you think there is a defect in the course materials, include the URL of the document you have a question about and a specific explanation of the defect or contradiction
  - Start your e-mail with "I've used the e-mail question checklist, and I think the following is an issue:" or the e-mail might not be replied to
  - Wait 5 minutes before sending. Seriously. We get lots of "oops, found it" e-mails less than 5 minutes after sending a query

Project 5 - Overview

- **Implementation first half of elevator**
  - Door Control
  - Drive Control
  - Car Button Control
  - Hall Button Control
- **Traceability - State chart to code**
- **Unit testing**
- **Integration testing**
- **Peer Review**



Implementation

- **Create new java files to implement four controllers**
  - Place these files in ../simulator/elevatorcontrol/
  - Each module must be included in simulator.elevatorcontrol package

- **General requirements listed on the website. Some examples:**
  - You shall use the interface defined in the behavioral requirements
  - You shall NOT add additional communication channels between controllers
    - No accessing global variables, etc.
    - Just communicate using network and physical messages
  - You shall adhere to the message dictionary and interface
    - Don’t be tempted to create new messages or modify the dictionary

- **We’ll eventually run your implementations on our own test files**
  - Probably fail tests if your design uses secondary channels or altered dictionary

Traceability

- **All transition arcs must be traced to the code that causes the transition**
  - In most cases, comment just above the if statement that tests guard statement
- **Code must contain comments that indicates each transition**
  - Forward traceability
- **Portfolio must include traceability table**
  - Each transition and its corresponding code line # must be in the table
  - Backward traceability
- **Detailed instructions and hints on project 5 web page**

Testing

- **Project 5 page contains link to detailed instructions for testing**
  - You must perform each step listed in the detailed testing instructions
- **Unit Tests**
  - Exercise all the transitions in your state chart
  - Reminder: If your transition has an OR, you must test both branches!
- **Integration Tests**
  - Select TWO sequence diagrams
    - Shall include *at least* one of the implemented modules
    - Should NOT include *any* of the non-implemented modules
- **You are not required to pass every test**
  - You shall document the results of every test
- **Traceability required for each test**
- **Peer review required for each Unit Test**

Simulator Documentation

- **There is LOTS of documentation. (Believe me. ☺)**
- **Spend some time getting familiar with it!**
- **Codebase page on the course website**
  - http://www.ece.cmu.edu/~ece649/project/codebase/index.html

- **Javadoc**
  - Describes simulator classes in detail
  - How to build simulator javadoc:
    - Download the latest version of the simulator
    - Run ‘make’ in the top-level directory (not the code directory)
    - This creates a folder called ‘doc’ with the javadoc for the simulator
  - Javadoc is mostly up to date, but may contain some references to outdated simulator

Simulator Documentation

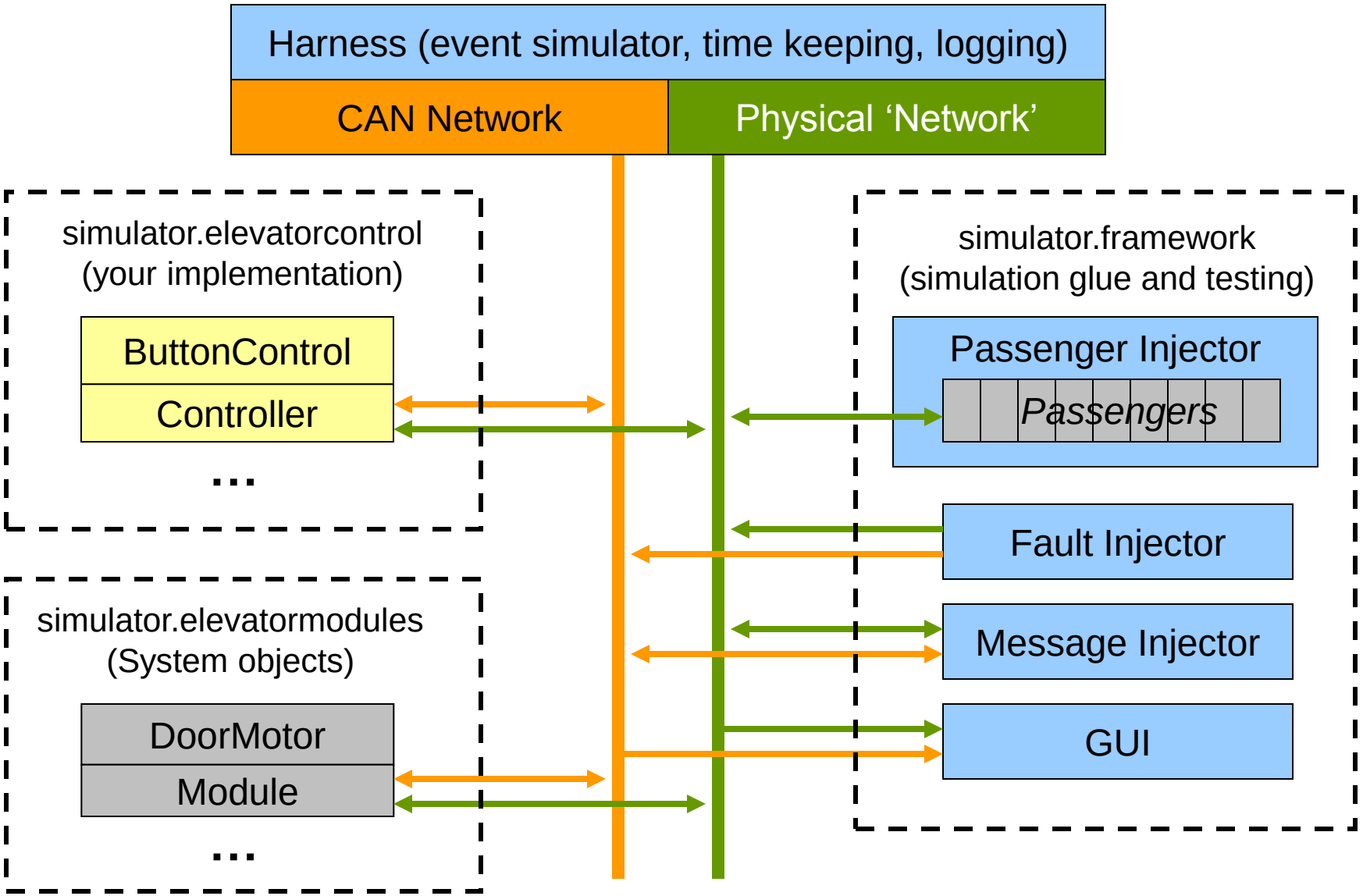
- **Command line interface**
  - Run simulator with no arguments
  - Read it! Lots of useful details and features!

- **Examples**
  - Check the provided example code if you having trouble getting your interfaces or tests working.
  - Testlight, soda machine example

Code Commenting Style

- **Simulator development overview has a complete style guide**
- **Traceability comments *shall*** be **exactly as specified in the project**
- **Other guidelines are recommendations, not hard and fast rules**
- **Your code *shall* be easily understood by a reasonable third party**
  - For example, the TAs
- **If in doubt, refer to the examples or come to office hours**

Simulator Architecture



Controller Implementation

- **All controllers must be a descendent of** simulator.framework.Controller
- **Provides CAN network and physical interfaces**
  - Enforces rules on like “only one physical input” and “only one physical output”
- **Provides a timer object (for periodic execution)**
- **Provides logging framework**
  - See Simulator Debug Tips on course webpage

Physical ‘Network’ Interface

- **Controller provides a** PhysicalConnection **object**

- **Important Methods**
  - registerTimeTriggered(Payload object)
  - sendTimeTriggered(Payload object, SimTime period)

- **registerTimeTriggered(Payload object)**
  - a.k.a. physical input
  - The payload object will be updated periodically with current value.

- **sendTimeTriggered(Payload object, SimTime period)**
  - a.k.a. physical output
  - When you modify the value in the payload object, that modification will be periodically propagated to the rest of the system.
  - Period should be the same as the controller period.

CAN Network Interface

- **Controller provides a** CANConnection **object**

- **Important Methods**
  - registerTimeTriggered(CanMailbox object)
  - sendTimeTriggered(CanMailbox object, SimTime period)

- **registerTimeTriggered(CanMailbox object)**
  - a.k.a. network input
  - Mailbox object updated whenever a CAN message with the same ID is sent

- **sendTimeTriggered(CanMailbox object, SimTime period)**
  - a.k.a. network output
  - A CAN message is sent periodically
    - Message has whatever value is currently in the mailbox object
  - Period should be the same as the controller period.

CAN Network Translators

- **Physical payload objects have field that represents the data value**
  - E.g. ‘CarCallPayload.pressed’
- **CanMailbox objects only have bit-level representation of CAN message**
  - Can store up to 8 bytes of data per message, per the CAN spec
- **Use CanPayloadTranslators to convert bit sets into abstract ‘get/set’ methods**
  - Examples provided in the codebase
  - You can write your own or use the ones provided
  - Use consistent translators
    - Sender and receiver of same message must use same translator
  - **Translators are also used in the testing framework**

Testing Framework

- **The -mf and -cf file formats are fully documented in the command line documentation**
  - Read the documentation carefully
  - Make sure your text files have unix line endings
  - You can also look at the TestLight examples from project 1

- **-cf <file.cf> to specify which objects should be instantiated**
  - Test a single object (unit test)
  - or a set of objects (integration test)

- **-mf <file.mf> to define the test inputs and outputs**
  - Inputs - inject CAN messages and physical/framework values
  - Outputs - use assertions to monitor controller outputs

- **Run the simulator with no args to see info about the file syntax**

Testing Framework

- **The message injector has a simple macro feature for –mf files**
  - Syntax: #DEFINE MACRO value
  - Macro is a one-for-one text field replacement
  - Cannot be used to replace multiple fields
- **Use macros for things that are subject to change**
  - CAN message IDs
  - Message periods
- **-pd to generate and exhaustive list of #define statements**
  - Save output to file, then #INCLUDE in your test files
- **Use descriptive macro names to improve readability**
  - See soda machine examples

Testing Framework Tips

- **'F'/Framework is a synonym for the physical network**
  - http://www.ece.cmu.edu/~ece649/project/sodamachine/portfolio/unit_test/button_control_1.mf
- **Invalid test file lines can cause cryptic runtime errors**
- **A good workflow for defining tests is:**
  - Most unit tests only use a handful of inputs and outputs
  - For each test, start out with just one injection line for each input and one assertion for each output
  - Run the test until you have syntax correct (get no errors)
  - Use the validated lines as models for the rest of the test
- **Start your testing early!!!**
  - Testing takes a long time, do not blow it off until the last minute

Testing Framework Tips

- **<period> parameter specifies how often the message is sent**
  - use the periods defined in Control.java and Modules.java
  - Once you start using a period value for a message, you cannot change the period later in the test

- **<time> parameter specifies what time in the simulator a message change occurs**
  - For periodic messages, change is rounded to next time when the message is sent

Soda Machine Example

- **It's all there:**
  - Code (java)
  - Testing
    - .mf, .cf files
    - Sequence Diagrams
  - Traceability

Questions?